readme

design and implementation

This project was implemented as a program written in both C and x86 assembly. The C code handles the "wrapper" functions, like usage requirements, detection of bad input (negative numbers) and interpretation of output information (answer or overflow). The function to calculate the actual Fibonacci number is written in assembly. There are instructions to return 1 if the value is less than 2, and otherwise continues to a loop, after initializing registers for use as variables.

challenges

Attempting to use the loop instruction proved complicated, as my algorithm counts "up" to the sent value rather than "down" like the loop instruction does on its own.
I had trouble implementing this requirement, and instead submitted an assignment that worked well and reasonably similarly, just using jump instructions and my own counter register, and repurposing the register loop would normally use.
I did not realize this was a requirement until too late.

analysis (big-o) of space and time performance

If the number is negative, the c code notifies the user in constant time, consuming very little space in memory.
If the number sent to the function is less than 2, the answer (1) is returned in constant time, consuming very little space in memory.
When the loop is initiated, the answer is calculated in linear time, based on the value given, and as it operates, consumes a constant amount of space for storage of value while calculating.